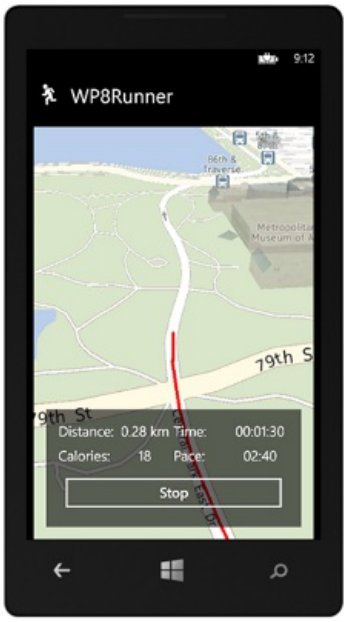
A Windows Phone 8 Run Tracking App in 100 Lines of Code!

This article explains how to create a run-tracker app using Windows Phone 8

Introduction

Windows Phone 8 brings a fantastic selection of new features for application developers. For a great overview see the Nokia Developer page on What's new in Windows Phone 8.

I have worked with Windows Phone 7 since it was in beta, so as you can imagine, I downloaded the Windows Phone 8 SDK as soon as it went live. For a bit of fun I decided to create a simple run-tracking application that showcases a number of these features ... and for an extra challenge do it all within 100 lines of code! (without resorting to writing compact and cryptic code).



This article guides you through the application that I developed, delving into the following Windows Phone 8 features:

- The new map control, with pedestrian and landmark features.
- How to track the users location, including tracking their location in the background while other apps are running
- Adding line annotations to the map
- Some 3D mapping features, setting the pitch and heading
- Live-tiles making use of one of the new tile formats

Whilst it was perfectly possible to develop a run-tracking app with Windows Phone 7 (and there are a number of good examples in the Marketplace), the new features and capabilities of Windows Phone 8 can be used to make a much more feature-rich application.

The Application User Interface

This application has quite a basic UI, which is composed of full-screen map, which has the run statistics overlayed on top of it as shown in the screenshot below:

The application UI is defined in XAML as follows:

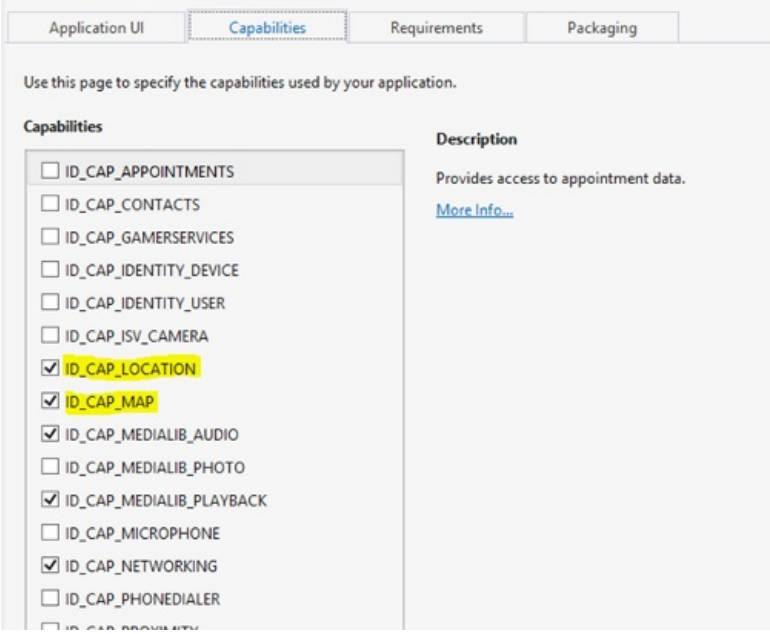
```
<Grid util:GridUtils.RowDefinitions="Auto, *">

  <!-- title -->
  <StackPanel Grid.Row="0" Margin="12,17,0,28">
    <StackPanel Orientation="Horizontal">
      <Image Source="/Assets/ApplicationIconLarge.png" Height="50"/>
      <TextBlock Text="WP8Runner" VerticalAlignment="Center"
                 Margin="10 0 0 0"
                 FontSize="{StaticResource PhoneFontSizeLarge}"/>
    </StackPanel>
  </StackPanel>

  <!--ContentPanel - place additional content here-->
  <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">

    <!-- the map -->
    <maps:Map x:Name="Map"
          ZoomLevel="16"/>

    <!-- run statistics -->
    <Grid Background="#99000000" Margin="20"
          VerticalAlignment="Bottom">
      <Grid Margin="20"
            util:GridUtils.RowDefinitions="40, 40, Auto"
            util:GridUtils.ColumnDefinitions="*, *, *, *">

        <!-- distance -->
        <TextBlock Text="Distance:"/>
        <TextBlock Text="0 km" Grid.Column="1" x:Name="distanceLabel"
              HorizontalAlignment="Center"/>
```

```
        <!-- time -->
        <TextBlock Text="Time:" Grid.Column="2"/>
        <TextBlock Text="00:00:00" Grid.Column="3" x:Name="timeLabel"
              HorizontalAlignment="Center"/>

        <!-- calories -->
        <TextBlock Text="Calories:" Grid.Row="1"/>
        <TextBlock Text="0" Grid.Column="1" x:Name="caloriesLabel"
              HorizontalAlignment="Center" Grid.Row="1"/>

        <!-- pace -->
        <TextBlock Text="Pace:" Grid.Column="2" Grid.Row="1"/>
        <TextBlock Text="00:00" Grid.Column="3" x:Name="paceLabel"
              HorizontalAlignment="Center" Grid.Row="1"/>

        <Button Content="Start"
                Grid.Row="2" Grid.ColumnSpan="4"
                Click="StartButton_Click"
                x:Name="StartButton"/>
      </Grid>
    </Grid>
  </Grid>
</Grid>
```

`GridUtils` is a utility class, which I wrote a number of years ago, that provides convenient shorthand for defining grid columns and rows (for WPF, Silverlight and WindowsPhone). If you are following along, by building this running app from scratch, then in order to add the map, you will have to include the following namespace definition:

```
xmlns:maps="clr-namespace:Microsoft.Phone.Maps.Controls;assembly=Microsoft.Phone.Maps"
```

Before building and running the application, you have to include the mapping 'capability'. To do this open up **WPAppManifest.xml**, navigate to the Capabilities tab and check the `ID_CAP_MAP` checkbox. While you're there, you may as well include `ID_CAP_LOCATION` as well:



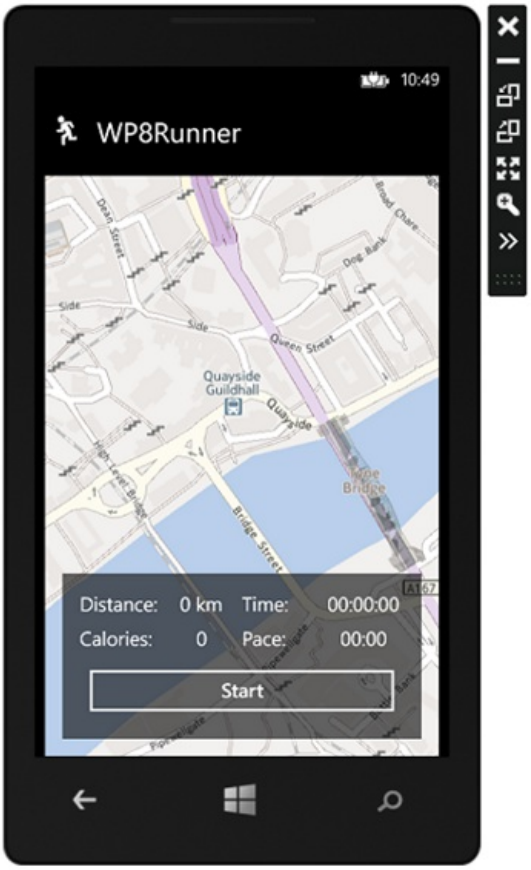
Capabilites are used to determine the phone features that your application uses so that users can more easily determine what an application does.

With these capabilities included, build and run the application and you should see the same UI that was illustrated above.

One of the improvements in the maps control is that it is fully vector-based (The Windows Phone 7 map is image-tile-based), this creates a much more smooth transition when the map is zoomed, and also allows for 3D transformations (as we will see a little later on). The map control also has a few other useful features for our running app, pedestrian-features and landmarks. These can be enabled as follows:

```
<!-- the map -->
<maps:Map x:Name="Map"
      PedestrianFeaturesEnabled="True"
      LandmarksEnabled="True"
      ZoomLevel="16"/>
```

With these features enabled the map illustrates useful features such as stairs, crossings and 3D landmarks:



(By the way, I'm not counting the ~50 lines of XAML in my total lines-of-code count!)

The Windows Phone 8 maps have many more new features that I have not used in this application. You could for example use the new `ColorMode`, which allows you to render a 'dark' map which is easier on the eyes in low light conditions. You could even make the run-tracking app choose the `ColorMode` based on the time of day!

Timing The Run

When the **Start** button is tapped the application tracks the user's location using the phone's built in GPS receiver, in order to mark their path on the map. It also times their run duration and generates various statistics of interest. We'll start with the simpler of the two, timing the run. When the start button is clicked a `DispatcherTimer` is started and the time of the button tap recorded. On each timer 'tick' the label which indicates the elapsed run time is updated:

```
public partial class MainPage : PhoneApplicationPage
{
  private DispatcherTimer _timer = new DispatcherTimer();
  private long _startTime;

  public MainPage()
  {
```

```
    InitializeComponent();

    _timer.Interval = TimeSpan.FromSeconds(1);
    _timer.Tick += Timer_Tick;
  }

  private void Timer_Tick(object sender, EventArgs e)
  {
    TimeSpan runTime = TimeSpan.FromMilliseconds(System.Environment.TickCount -
_startTime);
    timeLabel.Text = runTime.ToString(@"hh\:mm\:ss");
  }

  private void StartButton_Click(object sender, RoutedEventArgs e)
  {
    if (_timer.IsEnabled)
    {
      _timer.Stop();
      StartButton.Content = "Start";
    }
    else
    {
      _timer.Start();
      _startTime = System.Environment.TickCount;
      StartButton.Content = "Stop";
    }
  }
}
```

With the above code in place, tapping the **start** button starts the timer.

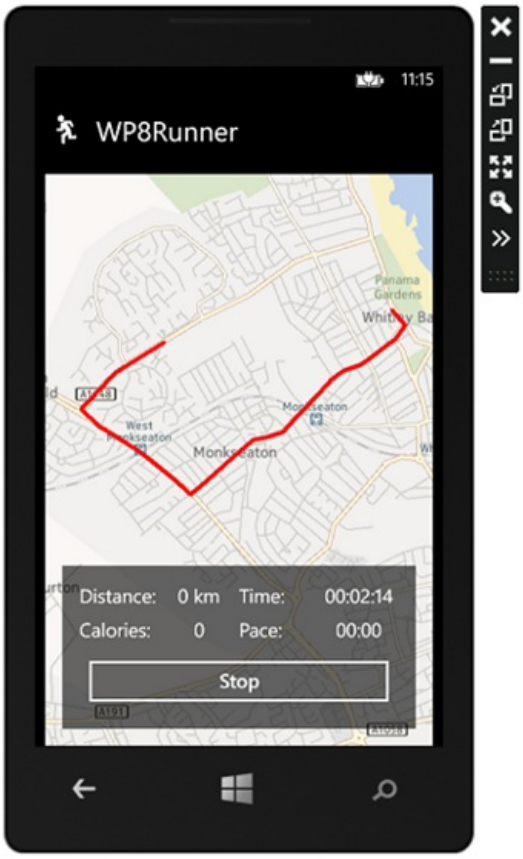
Location Tracking

The next step is to track the location whilst the timer is running. The Windows Phone API has a `GeoCoordinateWatcher` class which fires a `PositionChanged` event which can be used to track the user's location. It is very easy to render the user's movements on a map via a `MapPolyLine`, which is a line path which is defined in terms of `geocoordinates`. Each time the event is fired, a new point is added to the line as follows:

```
public partial class MainPage : PhoneApplicationPage
{
  private GeoCoordinateWatcher _watcher = new
GeoCoordinateWatcher(GeoPositionAccuracy.High);
  private MapPolyline _line;
  private DispatcherTimer _timer = new DispatcherTimer();
  private long _startTime;

  public MainPage()
  {
    InitializeComponent();

    // create a line which illustrates the run
    _line = new MapPolyline();
    _line.StrokeColor = Colors.Red;
    _line.StrokeThickness = 5;
    Map.MapElements.Add(_line);
```

```
    _watcher.PositionChanged += Watcher_PositionChanged;

    //.. timer code omitted ...
  }

  //.. timer code omitted ...

  private void StartButton_Click(object sender, RoutedEventArgs e)
  {
    if (_timer.IsEnabled)
    {
      _watcher.Stop();
      _timer.Stop();
      StartButton.Content = "Start";
    }
    else
    {
      _watcher.Start();
      _timer.Start();
      _startTime = System.Environment.TickCount;
      StartButton.Content = "Stop";
    }
  }


  private void Watcher_PositionChanged(object sender,
GeoPositionChangedEventArgs<GeoCoordinate> e)
  {
    var coord = new GeoCoordinate(e.Position.Location.Latitude,
e.Position.Location.Longitude);

    Map.Center = coord;
    _line.Path.Add(coord);
  }
}
```

With these few lines of extra code, the path of the user's run is added to the map:



The `PositionChanged` event handler can be developed further to compute the total run distance, calories burnt and pace. This makes use of the `GeoCoordinate.GetDistanceTo` method which can be used to compute the distance between two locations:

```
private double _kilometres;
private long _previousPositionChangeTick;

private void Watcher_PositionChanged(object sender,
GeoPositionChangedEventArgs<GeoCoordinate> e)
{
  var coord = new GeoCoordinate(e.Position.Location.Latitude,
e.Position.Location.Longitude);

  if (_line.Path.Count > 0)
  {
    // find the previos point and measure the distance travelled
    var previousPoint = _line.Path.Last();
    var distance = coord.GetDistanceTo(previousPoint);

    // compute pace
    var millisPerKilometer = (1000.0 / distance) * (System.Environment.TickCount -
_previousPositionChangeTick);

    // compute total distance travelled
    _kilometres += distance / 1000.0;

    paceLabel.Text = TimeSpan.FromMilliseconds(millisPerKilometer).ToString(@"mm\:ss");
    distanceLabel.Text = string.Format("{0:f2} km", _kilometres);
    caloriesLabel.Text = string.Format("{0:f0}", _kilometres * 65);
  }


  Map.Center = coord;
```

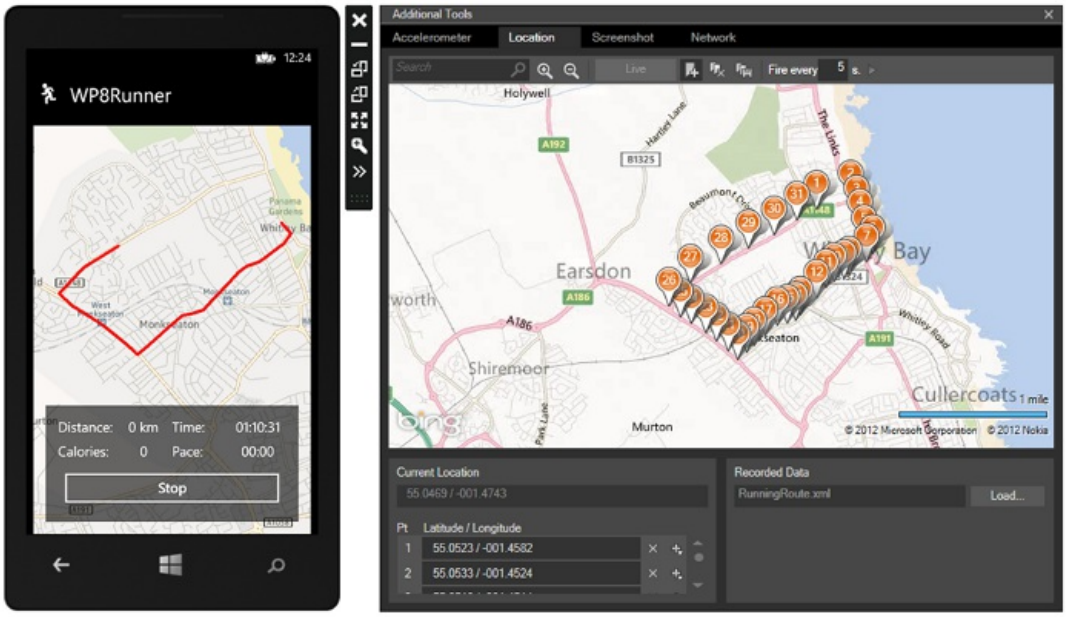


```
    _line.Path.Add(coord);
    _previousPositionChangeTick = System.Environment.TickCount;
}
```

Runner's do not measure pace in miles or kilometers per hour. Instead, pace is measured in terms of the time taken to travel a set distance. This method of measurement makes it much easier to determine your overall race time, e.g. if you are running at 4:00 minute-kilometers pace, you will complete a 5k race in 20 minutes.

NOTE: The code above uses a pretty basic calorie calculation, assuming a burn rate of 65 calories per kilometer. A more accurate calculation would incorporate the runner's weight and pace, and other environmental factors. I'll leave this as an exercise for the reader!

For developing applications that involve tracking a user's location the emulator has some very useful features. You can record points along a route, then replay them at set intervals. You can also save the route as an XML file so that it can be replayed in future sessions:



It takes a while to create a realistic dataset that emulates a real run, but at least you only have to do this once!

Setting The Map Pitch and Heading

Because of the vector nature of the Windows Phone 8 map it is possible to transform the view using the Pitch and Heading properties. The Pitch property sets the viewing angle of the map, providing a perspective rendering, rather than a top-down rendering, while the Heading property allows you to rotate the map. Most sat-nav systems use a combination of these effects to render the map so that it looks the same as the view directly in front of you. Many people find this type of map view much easier to understand (they do not have to perform rotate transforms in their head!).

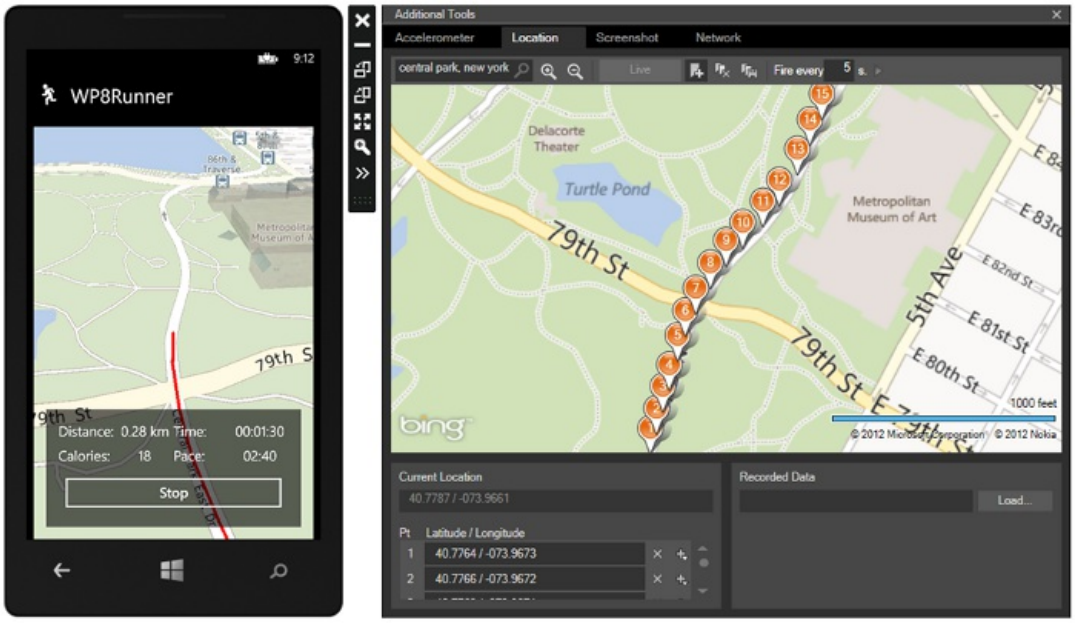
Adding this feature to the running app is really easy, firstly setting the map Pitch is simply done in the XAML:

```
<!-- the map -->
<maps:Map x:Name="Map"
      PedestrianFeaturesEnabled="True"
      LandmarksEnabled="True"
      Pitch="55"
      ZoomLevel="18"/>
```

Computing the heading is a little more complicated. In the previous section the current and previous location was used to compute pace and distance traveled. These two locations can be used to compute the heading, although the calculation is a little more involved. Fortunately I found a .NET library that contains some useful geolocation utilities, including one that computes heading. Using the .NET Extra library, finding and setting the heading is quite straightforward:

```
PositionHandler handler = new PositionHandler();
var heading = handler.CalculateBearing(new Position(previousPoint), new
Position(coord));
Map.SetView(coord, Map.ZoomLevel, heading, MapAnimationKind.Parabolic);
```

Also, note that the above code uses the map `SetView` method rather than setting each property independently. If you set the properties directly, the map state changes immediately, which means that the view will 'jump' from one location/heading to another. Whereas `SetView` transitions from one location to another, producing a much more fluid UI. You can see the use of heading and pitch below, with a run in New York's Central Park:



Background Location Tracking

With Windows Phone 7 you could run the foreground applications under the lock screen, which is a pretty important feature for a sports tracking application, this allows the user to lock their phone while their location is still tracked. Windows Phone 8 goes one better, applications that track geolocation can run in the background, which means they can keep tracking the user's location while they use other apps – checking emails, or playing music for example.

In order to turn on this feature you have to edit **WMAppManifest.xml** by hand, to do this right-click the file and select **View code**. Then locate the `Tasks` element and add the following.

```
<Tasks>
  <DefaultTask Name="_default" NavigationPage="MainPage.xaml">
    <BackgroundExecution>
      <ExecutionType Name="LocationTracking" />
    </BackgroundExecution>
  </DefaultTask>
</Tasks>
```

And that's it!

When the application starts running in the background, the `RunningInBackground` application event is fired. You could use this to show a toast notification for example, but in the next section we'll look at a more interesting way of keeping the user informed of the continued location tracking.

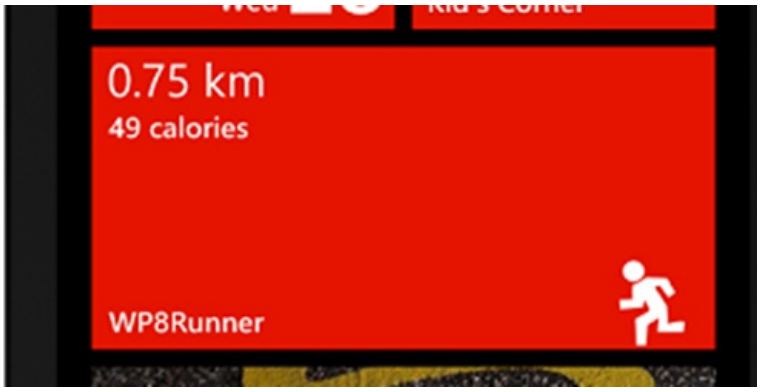
A Live Tile

Windows Phone 8 adds yet more tile templates, we'll use the new 'Iconic Template' here. To select the template open up **WMAppManifest.xml** (using the visual editor this time!), and select the TemplateIconic template.

Updating the tile state is as simple as sending a notification. Each time the location changes the following code is executed:

```
ShellTile.ActiveTiles.First().Update(new IconicTileData()
{
  Title = "WP8Runner",
  WideContent1 = string.Format("{0:f2} km", _kilometres),
  WideContent2 = string.Format("{0:f0} calories", _kilometres * 65),
});
```

Now if you pin the application to the start screen and use a wide tile format, while the location is being tracked in the background, the tile updates:



And with that final change addition, the running application is complete!

Conclusion

Windows Phone 8 has some pretty cool new features that allow you to extend the capabilities of your application. In this article I have shown how a simple run tracking app benefits from many of these new features. Also, the expressive power of the APIs and frameworks allow you to develop complex applications with very little code.

Clearly the application illustrated here is not complete! Why not have a go at developing it further yourself - why not try using isolated storage for recording your run history? or add summary statistics as charts? You could try using some of the other new Windows Phone 8 APIs such as voice-commands to control the start / stop of each run? Have fun!

Source code

So, is this app really just 100 lines of code? You can download the sourcecode (File:WP8RunnerSource.zip) and see for yourself that **MainPage.xaml.cs** is exactly 100 lines.